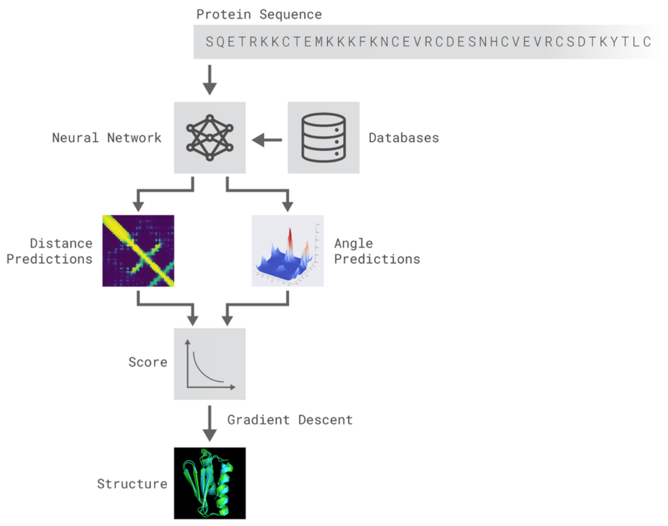
Notebook 1: First Data preprocessing and Cleaning

Quick Introduction:

- The Protein Folding Problem (predicting a protein structure from its sequence) is an interesting one since DNA sequence data available is becoming cheaper and cheaper at an unprecedented rate. Recent research has applied Deep Learning techniques in order to accurately predict the structure of polypeptides.
- In this presentation, I will present an attempt to imitate the AlphaFold system for protein prediction architecture.
  - We use 1-D Residual Networks (ResNets) to predict dihedral torsion angles (no more technical details on this, just enough to recall that adjacent amino acids in a protein chain bond together NOT in a linear manner)(remember interactions between non-adjacent amino acids).
  - We use the CASP7 (one of the competitions) ProteinNet dataset (resource below) for training and evaluation of the model.
  - We only train and predict aangles of proteins with less than 200 AAs. No larger proteins nor crops of larger proteins are used.
  - Finally, we compare the results with those obtained by AlphaFold, and mention some insights on why AlphaFold's approach is the better suited for the problem.



(Image from DeepMind's original blogpost.)

Configuring Julia on Jupyter Notebook (Skip)

In [ ]:
```
# # Using Julia on Jupyter Notebook:

# %%shell
# set -e

# #---------------------------------------------------#
# JULIA_VERSION="1.0.0" # any version ≥ 0.7.0
# JULIA_PACKAGES="IJulia BenchmarkTools PyCall PyPlot"
# JULIA_PACKAGES_IF_GPU="CUDA" # or CuArrays for older Julia versions
# JULIA_NUM_THREADS=4
# #---------------------------------------------------#

# if [ -z `which julia` ]; then
#   # Install Julia
#   JULIA_VER=`cut -d '.' -f -2 <<< "$JULIA_VERSION"`
#   echo "Installing Julia $JULIA_VERSION on the current Colab Runtime..."
#   BASE_URL="https://julialang-s3.julialang.org/bin/linux/x64"
#   URL="$BASE_URL/$JULIA_VER/julia-$JULIA_VERSION-linux-x86_64.tar.gz"
#   wget -nv $URL -O /tmp/julia.tar.gz # -nv means "not verbose"
#   tar -x -f /tmp/julia.tar.gz -C /usr/local --strip-components 1
#   rm /tmp/julia.tar.gz

#   # Install Packages
#   nvidia-smi -L &> /dev/null && export GPU=1 || export GPU=0
#   if [ $GPU -eq 1 ]; then
#     JULIA_PACKAGES="$JULIA_PACKAGES $JULIA_PACKAGES_IF_GPU"
#   fi
#   for PKG in `echo $JULIA_PACKAGES`; do
#     echo "Installing Julia package $PKG..."
#     julia -e 'using Pkg; pkg"add '$PKG'; precompile;"' &> /dev/null
#   done

#   # Install kernel and rename it to "julia"
#   echo "Installing IJulia kernel..."
#   julia -e 'using IJulia; IJulia.installkernel("julia", env=Dict(
#       "JULIA_NUM_THREADS"=>"'"$JULIA_NUM_THREADS"'"))'
#   KERNEL_DIR=`julia -e "using IJulia; print(IJulia.kerneldir())"`
#   KERNEL_NAME=`ls -d "$KERNEL_DIR"/julia*`
#   mv -f $KERNEL_NAME "$KERNEL_DIR"/julia

#   echo ''
#   echo "Successfully installed `julia -v`!"
#   echo "Please reload this page (press Ctrl+R, ⌘+R, or the F5 key) then"
#   echo "jump to the 'Checking the Installation' section."
# fi
```

```
Installing Julia 1.0.0 on the current Colab Runtime...
2023-01-03 21:11:17 URL:https://storage.googleapis.com/julialang2/bin/linux/x64/1.0/julia-1.0.0-linux-x86_64.tar.gz [88896624/8
8896624] -> "/tmp/julia.tar.gz" [1]
Installing Julia package IJulia...
Installing Julia package BenchmarkTools...
Installing Julia package PyCall...
Installing Julia package PyPlot...
Installing IJulia kernel...
[ Info: Installing julia kernelspec in /root/.local/share/jupyter/kernels/julia-1.0

Successfully installed julia version 1.0.0!
Please reload this page (press Ctrl+R, ⌘+R, or the F5 key) then
jump to the 'Checking the Installation' section.
```

Out[5]:

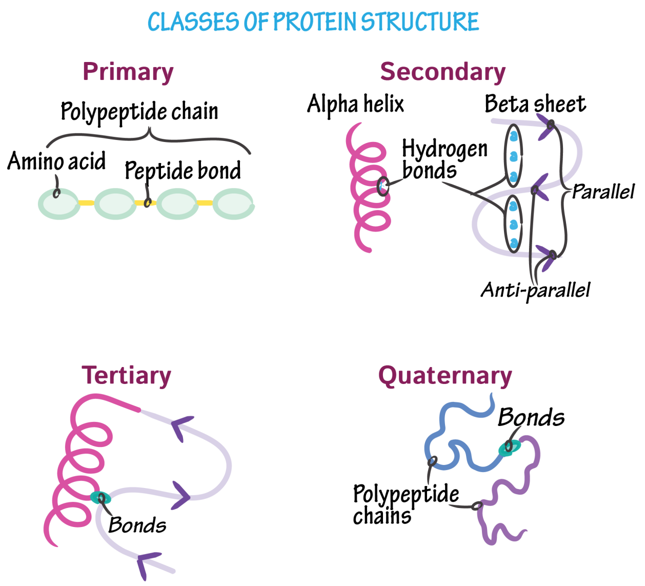
Preprocessing Data:

- **CASP7:**
  - Resource: https://sharehost.hms.harvard.edu/sysbio/alquraishi/proteinnet/human_readable/casp7.tar.gz (https://sharehost.hms.harvard.edu/sysbio/alquraishi/proteinnet/human_readable/casp7.tar.gz)
- **About data:**
  - 4 pieces of data given per protein:
    - id
    - class of protein and the amino acid sequence
    - space coordinates of amino acids
    - ppsm (Position Specific Scoring Matrix)
- **Aims now:**
  - Preprocess the raw data file and handle it properly.
  - Remove unnecessary data.
  - Reduce data file from 600mb to 170mb.
  - Choose proteins that are composed of less than 200 amino acids.

- **Raw data:**

```
[ID]
1VBK_1_A
[PRIMARY]
MNVVIVRYGEIGTKSRQTRSWFEKILMNNIREALVTEEVPYKEIFSRHGRIIVKTNSPKEAANVLVRVFGIVSISPAMEVEASLEKINRTALLMFRKKAKEVGKERPKFRVTARRITKEFPLDSLEIQAKVGEYILNNENCEVDLKNYDIEIGI
[EVOLUTIONARY]
0	0	0.037827352085354024	0.006188757091284167	0	0.0626517727051967	0	0.031385803959439885	0.04104297440849831	0	0
1455005	0.18539786710418374	0.07272727272727271	0.21261777959852518	0.07210159770585826	0.1306306306306306	0.007774140752864157
16443 0.06001558846453624	0	0.06332665330661322	0	0	0	0.023246492985971947	0.04727564102564103	0.02846832397754611	0.14
330494037478705	0.1954022988505747	0	0.18398637137989782	0	0	0	0.01447424435930183	0	0	0.016602809706257982	0.07
2	0.05365642578970144	0.03713298791018997	0.09122203098106715	0.07986260197509659	0.01230899830220713	0.06186478126380909	0.05
0	0	0.01842870999030068	0	0	0.009227780475959202	0	0	0	0	0	0	0.004312410158121706	0	0	0	0
803364036844213	0.01162324649298597	0.013627254509018034	0.006412825651302605	0.010971149939049166	0	0	0.044632086851628464	0.00
640067911714771	0	0	0	0.11738351254480288	0	0	0	0	0	0.18885987815491734	0	0	0.18910675381263617	0
0	0.29824561403508776	0	0	0	0	0	0	0.013037180106228875	0.009169884169884169	0	0	0	0	0	0.04882719
771543086172	0	0	0.09375	0	0	0.008821170809943863	0.0825320512820513	0.011231448054552748	0.0092110532639167	0.00
27865168539325847 0.06861443116423196	0	0	0	0	0	0	0	0	0	0.014621178555604786	0.01993797075764289	0.11456558
022687609075043632	0	0	0	0	0	0	0	0	0.028733130169786673	0	0.016986062717770038	0	0	0	0.1503923
0	0.033138401559454196	0.04267701260911737	0	0	0	0	0	0	0.9908301158301158	0	0	0	0	0	0.08808042
72306 0.26893787575150296	0	0.2539046856227473	0.03877221324717286	0.11438965238480193	0.027655310621242483	0.34723336006415395
307	0	0.1270096463022508	0.05913113435237329	0	0.23049074818986323	0.004825090470446321	0.24165661439485328	0	0.03297145
75849 0	0	0.09750566893424037	0.01111699311805188	0.3383838383838384	0.05203503348789284	0	0.26774053112755763	0.2556263
0	0	0	0.1964930376482723	0.011747430249632892	0	0	0.18928054080154516	0	0	0	0.10698795180722892	0	0
	0	0	0.07975951903807614	0	0.023255813953488372	0	0.02085004009623095	0.023637820512820516	0.04211793020457281	0.14
0	0	0.1660026560424967	0	0	0.033658104517271914	0	0	0	0	0	0	0	0	0.04560954816709292	0	0
156794425087108	0.16717457553330428	0.1236395298215063	0.02221254355400697	0	0	0	0	0	0.0509581881533101	0	0.00
0	0	0	0	0	0	0	0.007242877836793819	0.65620473201352	0	0	0.18650602409638556	0	0	0.46979865771812085
3	0.004844570044408559	0.02284569138276553	0	0.033640368442130565	0.03877221324717286	0.013742926434923199	0	0.012830793905371
01165594855305466 0.020514883346741754	0	0.014078841512469832	0	0	0	0.08604744672295937	0	0	0	0.020064205457463888
774	0	0.23424190800681435	0.22795057520238604	0.21920135938827523	0.015371477369769425	0.004486316733961418	0.0778611632270169
0	0	0.11154219204655674	0	0	0	0.0548810101991258	0.07967165620473202	0.009657170449058426	0	0	0	0	0
9	0.01002004008016032	0	0	0.006410256410256412	0.008420208500400962	0	0.02085004009623095	0.00841346153846154	0.06778981
```

Classes of Protein Structures:



In [26]:
```julia
# Julia's way of listing files in current directory:
println(readdir())
```

```
[".ipynb_checkpoints", "angle_data_preparation_py.ipynb", "get_angles_from_coords_py.ipynb", "julia_get_proteins_under_200aa.ip
ynb", "MiniFold", "predicting_angles.ipynb", "resnet_1d_angles.py"]
```

In [27]:
```julia
# Opening the training set (saved as a text file):
f = open("C://Users/beshe/OneDrive/Desktop/PHYS496/MiniFold/data/training_30.txt")
```

Out[27]:
```
IOStream(<file C://Users/beshe/OneDrive/Desktop/PHYS496/MiniFold/data/training_30.txt>)
```

In [28]:
```julia
# Reading file:
lines = readlines(f)
```

Out[28]:
```
340989-element Array{String,1}:
 "[ID]"
 "1VBK_1_A"
 "[PRIMARY]"
 "MNVVIVRYGEIGTKSRQTRSWFEKILMNNIREALVTEEVPYKEIFSRHGRIIVKTNSPKEAANVLVRVFGIVSISPAMEVEASLEKINRTALLMFRKKAKEVGKERPKFRVTARRITKEFPLD
SLEIQAKVGEYILNNENCEVDLKNYDIEIGIEIMQGKAYIYTEKIKGWGGLPIGTEGRMIGILHDELSALAIFLMMKRGVEVIPVYIGKDDKNLEKVRSLWNLLKRYSYGSKGFLVVAESFDRVL
KLIRDFGVKGVIKGLRPNDLNSEVSEITEDFKMFPVPVYYPLIALPEEYIKSVKERLGL"
 "[EVOLUTIONARY]"
 "0\t0\t0.037827352085354024\t0.006188757091284167\t0\t0.0626517727051967\t0\t0.031385803959439885\t0.04104297440849831\t0\t0
\t0.2221686746987952\t0.016770483948251078\t0\t0\t0.023934897079942556\t0.019197207678883076\t0.008111762054979722\t0\t0.0284
5709204090707\t0.013578624616732373\t0\t0.005000000000000001\t0.02\t0.03498542274052479\t0.057875155022736664\t0.157089706490
28522\t0.03760330578512397\t0.019016122364613478\t0.0070247933884297524\t0.04671351798263746\t0.08305785123966943\t0.26942148
76033058\t0.03223140495867768 4\t0.05378568473314026\t0.06657323055360898\t0.0748059280169372\t0.011235955056179777\t0.0533769
06318082784\t0.026435733819507746\t0.038358008075370126\t0.12897751378871447\t0.12587125871258711\t0.01804018040180402\t0.006
519967400162999\t0.0678861788617886\t0\t0.018570851836899473\t0.00404040404040404\t0.012111425111021397\t0.03754541784416633
\t0\t0.01332794830371567\t0.025030278562777557\t0.11465482438433588\t0.023415421881308032\t0.022983870967741942\t0.1019332161
6871705\t0.06119402985074626\t0.14236853356135099\t0.30067283431455005\t0.18539786710418374\t0.07272727272727271\t0.212617779
59852518\t0.07210159770585826\t0.1306306306306306\t0.007774140752864157\t0\t0.032714054927302096\t0.014129995962858296\t0.019
38610662358643\t0.13166397415185785\t0.035526846992329435\t0.08037156704361874\t0.07313131313131312\t0.02382875605815832\t0.3
```

In [29]:
```julia
# Defining our first function:

function coords_split(lister, splice)
    # Split all passed sequences by "splice" and return an array of them
    # Convert string fragments to float
    coords = []
    for c in lister
        push!(coords, [parse(Float64, a) for a in split(c, splice)])
    end
    return coords
end
```

Out[29]:
```
coords_split (generic function with 1 method)
```

In [30]:
```
# Scaning first n proteins:
# Organizing our proteins (classified into three types: Primary/Tertiary/Evolutionary):

names = []
seqs = []
coords = []
pssms = [] # Position Specific Scoring Matrix

# Record names, seqs, and coords for each protein btwn 1-n:
for i in 1:length(lines)
    if length(coords) == 995
        break
    end

    # Start recording
    if lines[i] == "[ID]"
        push!(names, lines[i+1])
    elseif lines[i] == "[PRIMARY]"
        push!(seqs, lines[i+1])
    elseif lines[i] == "[TERTIARY]"
        push!(coords, coords_split(lines[i+1:i+3], "\t"))
    elseif lines[i] == "[EVOLUTIONARY]"
        push!(pssms, coords_split(lines[i+1:i+21], "\t"))
        # Progress control
        if length(names)%50 == 0
            println("Currently @ ", length(names), " out of n")
        end
    end
end
```

```
Currently @ 50 out of n
Currently @ 100 out of n
Currently @ 150 out of n
Currently @ 200 out of n
Currently @ 250 out of n
Currently @ 300 out of n
Currently @ 350 out of n
Currently @ 400 out of n
Currently @ 450 out of n
Currently @ 500 out of n
Currently @ 550 out of n
Currently @ 600 out of n
Currently @ 650 out of n
Currently @ 700 out of n
Currently @ 750 out of n
Currently @ 800 out of n
Currently @ 850 out of n
Currently @ 900 out of n
Currently @ 950 out of n
```

In [31]:
```
# # Could use "Using LinearAlgebra + built-in norm()"
# function norm(vector)
#     return sqrt(sum([v*v for v in vector]))
# end
```

In [32]:
```
# Finding out how many proteins do we have with less than 200 amino acids among all proteins:

println("Total number of proteins: ", length(seqs))

n = 200
under = []
for i in 1:length(seqs)
    if length(seqs[i])<200
        push!(under, i)
        # println("Seelected with: ", length(seqs[i]), " number: ", i)
    end
end

println("Number of proteins under ", n, " : ", length(under))
```

```
Total number of proteins: 995
Number of proteins under 200 : 636
```

In [33]:
```
dists = []
# Get distances btwn pairs of AAs - only for prots under 200:
for k in under
    # Get distances from coordinates (Computing Norms using coordinates):
    dist = []
    for i in 1:length(coords[k][1])
        # Only pick coords for C-alpha carbons! - position (1/3 of total data)
        # i%3 == 2 Because juia arrays start at 1 - Python: i%3 == 1
        if i%3 == 2
            aad = [] # Distance to every AA from a given AA
            for j in 1:length(coords[k][1])
                if j%3 == 2
                    push!(aad, norm([coords[k][1][i],coords[k][2][i],coords[k][3][i]]-[coords[k][1][j],coords[k][2][j],coords[k]
                end
            end
            push!(dist, aad)
        end
    end
    push!(dists, dist)

    # Progress control
    if length(dists)%50 == 0
        println("Dists Currently @ ", length(dists), " out of n (500)")
    end
end
```

```
Dists Currently @ 50 out of n (500)
Dists Currently @ 100 out of n (500)
Dists Currently @ 150 out of n (500)
Dists Currently @ 200 out of n (500)
Dists Currently @ 250 out of n (500)
Dists Currently @ 300 out of n (500)
Dists Currently @ 350 out of n (500)
Dists Currently @ 400 out of n (500)
Dists Currently @ 450 out of n (500)
Dists Currently @ 500 out of n (500)
Dists Currently @ 550 out of n (500)
Dists Currently @ 600 out of n (500)
```

Now the proteins should be better organized and more accessible.

In [34]:
```
# Check everything's alright:
n = 2
println("id: ", names[n])
println("seq: ", seqs[n])
println("sample coord: ", coords[n][1][1])
println("sample dist: ", dists[n][1][5])
```

```
id: 2EUL_d2euld1
seq: MAREVKLTKAGYERLMQQLERERERLQEATKILQELMESSDDYDDSGLEAAKQEKARIEARIDSLEDILSRAVILEE
sample coord: 981.8
sample dist: 2674.450579090965
```

In [35]:
```
# Check another example (protein sequence):
n = 3
println("id: ", names[n])
println("seq: ", seqs[n])
println("sample coord: ", coords[n][1][1])
println("sample dist: ", dists[n][1][5])
```

```
id: 1QGV_1_A
seq: MSYMLPHLHNGWQVDQAILSEEDRVVVIRFGHDWDPTCMKMDEVLYSIAEKVKNFAVIYLVDITEVPDFNKMYELYDPCTVMFFFRNKHIMIDLGTGNNNKINWAMEDKQEMVDIIETVYRG
ARKGRGLVVSPKDYSTKYRY
sample coord: 0.0
sample dist: 1252.8694305473336
```

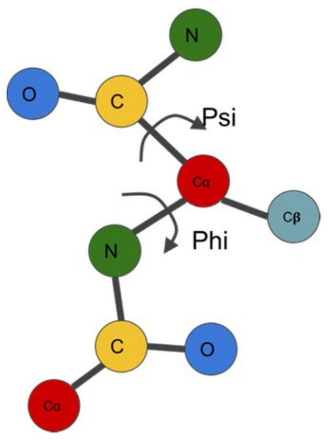
In [36]:
```
# Saving Data to a file:

using DelimitedFiles
open("C://Users/beshe/OneDrive/Desktop/PHYS496/MiniFold/data/full_under_200.txt", "a+") do f
    aux = [0]
    for k in under
        push!(aux, aux[length(aux)]+1)
        # ID
        write(f, "\n[ID]\n")
        write(f, names[k])
        # Seq
        write(f, "\n[PRIMARY]\n")
        write(f, seqs[k])
        # PSSMS
        write(f, "\n[EVOLUTIONARY]\n")
        writedlm(f, pssms[k])
        # Coords
        write(f, "\n[TERTIARY]\n")
        writedlm(f, coords[k])
        # Dists
        write(f, "\n[DIST]\n")
        # Check that saved proteins are less than 200 AAs
        if length(dists[aux[length(aux)]][1])>200
            print("error when checking protein in dists n: ", aux[length(aux)], " length: ", length(dists[aux[length(aux)]][1]))
            break
        else
            writedlm(f, dists[aux[length(aux)]])
        end
    end
end
```

Done!

In [ ]:
```
```

In [ ]:
```
```

Notebook 2: Calculate Dihedral Angles from Coordinates



In [14]:
```
# Import libraries - LOAD THE DATA
import numpy as np
import matplotlib.pyplot as plt
```

In [15]:
```
def parse_line(raw):
    return np.array([[float(x) for x in line.split("\t") if x != ""] for line in raw])
```

In [16]:
```
names = []
seqs = []
psis = []
phis = []
pssms = []
coords = []

path = "C://Users/beshe/OneDrive/Desktop/PHYS496/MiniFold/data/full_under_200.txt"
# Opn file and read text
with open(path, "r") as f:
    lines = f.read().split('\n')
```

In [17]:
```
# Extract numeric data from text:
for i,line in enumerate(lines):
    if len(names) == 601:
        break
    # Read each protein separately:
    if line == "[ID]":
        names.append(lines[i+1])
    elif line == "[PRIMARY]":
        seqs.append(lines[i+1])
    elif line == "[EVOLUTIONARY]":
        pssms.append(parse_line(lines[i+1:i+22]))
    elif line == "[TERTIARY]":
        coords.append(parse_line(lines[i+1:i+3+1]))
        # Progress control
        if len(names)%50 == 0:
            print("Currently @ ", len(names), " out of n")
```

```
Currently @  50  out of n
Currently @  100  out of n
Currently @  150  out of n
Currently @  200  out of n
Currently @  250  out of n
Currently @  300  out of n
Currently @  350  out of n
Currently @  400  out of n
Currently @  450  out of n
Currently @  500  out of n
Currently @  550  out of n
Currently @  600  out of n
```

In [18]:
```
# # Get the coordinates for 1 atom type:
# def separate_coords(full_coords, pos): # pos can be either 0(n_term), 1(calpha), 2(cterm)
#     res = []
#     for i in range(len(full_coords[1])):
#         if i%3 == pos:
#             res.append([full_coords[j][i] for j in range(3)])

#     return np.array(res)
```

In [19]:
```
# # Organize by atom type
# coords_nterm = [separate_coords(full_coords, 0) for full_coords in coords]
# coords_calpha = [separate_coords(full_coords, 1) for full_coords in coords]
# coords_cterm = [separate_coords(full_coords, 2) for full_coords in coords]
```

In [20]:
```
# # Check everything's ok
# print("Length coords_calpha: ", len(coords_cterm))
# print("Length coords_calpha[1]: ", len(coords_cterm[1]))
# print("Length coords_calpha[1][1]: ", len(coords_cterm[1][1]))
```

```
Length coords_calpha:  600
Length coords_calpha[1]:  142
Length coords_calpha[1][1]:  3
```

In [21]:
```
# Helper functions
def get_dihedral(coords1, coords2, coords3, coords4):
    """Returns the dihedral angle in degrees."""

    a1 = coords2 - coords1
    a2 = coords3 - coords2
    a3 = coords4 - coords3

    v1 = np.cross(a1, a2)
    v1 = v1 / (v1 * v1).sum(-1)**0.5
    v2 = np.cross(a2, a3)
    v2 = v2 / (v2 * v2).sum(-1)**0.5
    porm = np.sign((v1 * a3).sum(-1))
    rad = np.arccos((v1*v2).sum(-1) / ((v1**2).sum(-1) * (v2**2).sum(-1))**0.5)
    if not porm == 0:
        rad = rad * porm

    return rad
```

In [22]:
```
# Compute angles for a protein
phis, psis = [], [] # phi always starts with a 0 and psi ends with a 0
ph_angle_dists, ps_angle_dists = [], []
for k in range(len(coords)):
    phi, psi = [0.0], []
    # Use our own functions inspired from bioPython
    for i in range(len(coords_calpha[k])):
        # Calculate phi, psi
        # CALCULATE PHI - Can't calculate for first residue
        if i>0:
            phi.append(get_dihedral(coords_cterm[k][i-1], coords_nterm[k][i], coords_calpha[k][i], coords_cterm[k][i])) # my_calc

        # CALCULATE PSI - Can't calculate for last residue
        if i<len(coords_calpha[k])-1:
            psi.append(get_dihedral(coords_nterm[k][i], coords_calpha[k][i], coords_cterm[k][i], coords_nterm[k][i+1])) # my_calc

    # Add an extra 0 to psi (unable to claculate angle with next aa)
    psi.append(0)
    # Add protein info to register
    phis.append(phi)
    psis.append(psi)
```

```
C:\Users\beshe\AppData\Local\Temp\ipykernel_9424\3480274105.py:10: RuntimeWarning: invalid value encountered in true_divide
  v1 = v1 / (v1 * v1).sum(-1)**0.5
C:\Users\beshe\AppData\Local\Temp\ipykernel_9424\3480274105.py:12: RuntimeWarning: invalid value encountered in true_divide
  v2 = v2 / (v2 * v2).sum(-1)**0.5
```

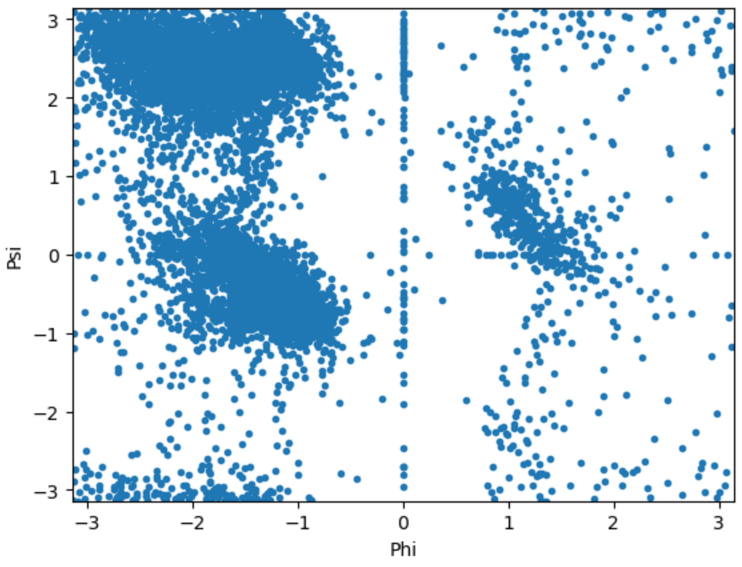
In [23]:
```
def stringify(vec):
    """ Helper function to save data to .txt file. """
    line = ""
    for v in vec:
        line = line+str(v)+" "
    return line

# Test function
print([stringify([1,2,3,4,5,6])])
```

```
['1 2 3 4 5 6 ']
```

In [24]:
```
# Check angles distribution is a Ramachandran Plot (2nd and 3rd quads. dense)
n = 100
test_phi = []
for i in range(n):
    for test in phis[i]:
        test_phi.append(test)
test_phi = np.array(test_phi)

test_psi = []
for i in range(n):
    for test in psis[i]:
        test_psi.append(test)
test_psi = np.array(test_psi)

# For quadrants following trigonometry positions
quads = [0,0,0,0]
for i in range(len(test_phi)):
    if test_phi[i] >= 0 and test_psi[i] >= 0:
        quads[0] += 1
    elif test_phi[i] < 0 and test_psi[i] >= 0:
        quads[1] += 1
    elif test_phi[i] < 0 and test_psi[i] < 0:
        quads[2] += 1
    else:
        quads[3] += 1

print("Quadrants: ", quads, " from ", len(test_phi))
```

```
Quadrants:  [542, 4459, 4458, 561]  from  10020
```

In [25]:
```
# Visualize data. Check it matches the Ramachandran Plot distribution
# (check if angles are well computed)
plt.scatter(test_phi, test_psi, marker=".")
plt.xlim(-np.pi, np.pi)
plt.xlabel("Phi")
plt.ylabel("Psi")
plt.ylim(-np.pi, np.pi)
plt.show()
```



In [28]:
```
# Data is OK. Can save it to file.
with open("C://Users/beshe/OneDrive/Desktop/PHYS496/MiniFold/data/full_angles_under_200.txt", "a") as f:
    for k in range(len(names)-1):
        # ID
        f.write("\n[ID]\n")
        f.write(names[k])
        # Seq
        f.write("\n[PRIMARY]\n")
        f.write(seqs[k])
        # PSSMS
        f.write("\n[EVOLUTIONARY]\n")
        for j in range(len(pssms[k])):
            f.write(stringify(pssms[k][j])+"\n")
        # PHI
        f.write("\n[PHI]\n")
        f.write(stringify(phis[k]))
        # PSI
        f.write("\n[PSI]\n")
        f.write(stringify(psis[k]))
```

Done!

Notebook 3: Preparing the Data

The data here prepared will be used to train the model.

About the Dataset to be prepared:

- The dataset here prepared will contain N proteins in a 21-dimensional representation
    - 20 dimensions for one-hot encoding + the Van der Waals radius of the AA.
- Plus 21 dimensions more for the PSSM (Position Specific Scoring Matrix)

In [17]:
```
# import libraries:
import numpy as np
import matplotlib.pyplot as plt
```

In [18]:
```
# Helper functions to extract numeric data from text:
def parse_lines(raw):
    return np.array([[float(x) for x in line.split(" ") if x != ""] for line in raw])

def parse_line(line):
    return np.array([float(x) for x in line.split(" ") if x != ""])
```

In [19]:
```
path = "C://Users/beshe/OneDrive/Desktop/PHYS496/MiniFold/data/full_angles_under_200.txt"
# Opn file and read text
with open(path, "r") as f:
    lines = f.read().split('\n')
```

In [20]:
```
# Scan first n proteins:
names = []
seqs = []
psis = []
phis = []
pssms = []

# Extract numeric data from text
for i,line in enumerate(lines):
    if len(names) == 601:
        break
    # Read each protein separately
    if line == "[ID]":
        names.append(lines[i+1])
    elif line == "[PRIMARY]":
        seqs.append(lines[i+1])
    elif line == "[EVOLUTIONARY]":
        pssms.append(parse_lines(lines[i+1:i+22]))
    elif lines[i] == "[PHI]":
        phis.append(parse_line(lines[i+1]))
    elif lines[i] == "[PSI]":
        psis.append(parse_line(lines[i+1]))
        # Progress control
        if len(names)%50 == 0:
            print("Currently @ ", len(names), " out of n")
```

```
Currently @  50  out of n
Currently @  100  out of n
Currently @  150  out of n
Currently @  200  out of n
Currently @  250  out of n
Currently @  300  out of n
Currently @  350  out of n
Currently @  400  out of n
Currently @  450  out of n
Currently @  500  out of n
Currently @  550  out of n
Currently @  600  out of n
```

In [21]:
```python
# Length of masking - 17x2 AAs
def onehotter_aa(seq, pos):
    pad = 17
    # Pad sequence
    key = "HRKDENQSYTCPAVLIGFWM"
    # Van der Waals radius
    vdw_radius = {"H": 118, "R": 148, "K": 135, "D": 91, "E": 109, "N": 96, "Q": 114,
                  "S": 73, "Y": 141, "T": 93, "C": 86, "P": 90, "A": 67, "V": 105,
                  "L": 124, "I": 124, "G": 48, "F": 135, "W": 163, "M": 124}
    radius_rel = vdw_radius.values()
    basis = min(radius_rel)/max(radius_rel)
    # Surface exposure
    surface = {"H": 151, "R": 196, "K": 167, "D": 106, "E": 138, "N": 113, "Q": 144,
               "S": 80, "Y": 187, "T": 102, "C": 104, "P": 105, "A": 67, "V": 117,
               "L": 137, "I": 140, "G": 0, "F": 175, "W": 217, "M": 160}
    surface_rel = surface.values()
    surface_basis = min(surface_rel)/max(surface_rel)
    # One-hot encoding
    one_hot = []
    for i in range(pos-pad, pos+pad): # alponer los guiones ya tiramos la seq para un lado
        vec = [0 for i in range(22)]
        # mark as 1 the corresponding indexes
        for j in range(len(key)):
            if seq[i] == key[j]:
                vec[j] = 1
                # Add Van der Waals relative radius
                vec[-2] = vdw_radius[key[j]]/max(radius_rel)-basis
                vec[-1] = surface[key[j]]/max(surface_rel)-surface_basis

        one_hot.append(vec)

    return np.array(one_hot)
```

In [22]:
```python
#Crops the PSSM matrix
def pssm_cropper(pssm, pos):
    pssm_out = []
    pad = 17
    for i,row in enumerate(pssm):
        pssm_out.append(row[pos-pad:pos+pad])
    # PSSM is Lx21 - solution: transpose
    return np.array(pssm_out)
```

In [23]:
```python
# Ensure all features relate to the same n. of prots
print("Names: ", len(names))
print("Seqs: ", len(seqs))
print("PSSMs: ", len(pssms))
print("Phis: ", len(phis))
print("Psis: ", len(psis))
```

```
Names:  600
Seqs:  600
PSSMs:  600
Phis:  600
Psis:  600
```

In [24]:
```python
input_aa = []
input_pssm = []
outputs = []
```

In [25]:
```python
long = 0 # Counter to ensure everythings fine

for i in range(len(seqs)):
    if len(seqs[i])>17*2:
        long += len(seqs[i])-17*2
        for j in range(17,len(seqs[i])-17):
        # Padd sequence
            input_aa.append(onehotter_aa(seqs[i], j))
            input_pssm.append(pssm_cropper(pssms[i], j))
            outputs.append([phis[i][j], psis[i][j]])
            # break
        # print(i, "Added: ", len(seqs[i])-34,"total for now:  ", long)
print("TOTAL:", long, len(input_aa))
```

```
TOTAL: 43001 43001
```

In [26]:
```
#Check everything's fine
print("Outputs: ", len(outputs))
print("Inputs AAs: ", len(input_aa))
print("Inputs PSSMs: ", len(input_pssm))
```

```
Outputs:  43001
Inputs AAs:  43001
Inputs PSSMs:  43001
```

Reshape the inputs

In [27]:
```
input_aa = np.array(input_aa).reshape(len(input_aa), 17*2, 22)
input_aa.shape
```

Out[27]:
```
(43001, 34, 22)
```

In [28]:
```
input_pssm = np.array(input_pssm).reshape(len(input_pssm), 17*2, 21)
input_pssm.shape
```

Out[28]:
```
(43001, 34, 21)
```

In [29]:
```
# Helper function to save data to a .txt file
def stringify(vec):
    return "".join(str(v)+" " for v in vec)
```

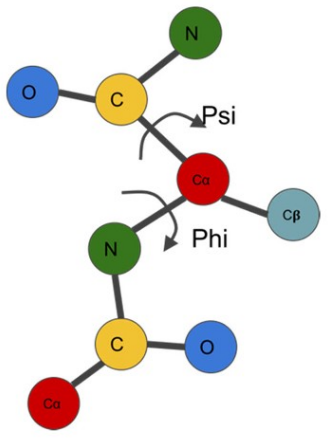
In [31]:
```
# Save outputs to txt file
with open("C://Users/beshe/OneDrive/Desktop/PHYS496/MiniFold/data/outputs.txt", "a") as f:
    for o in outputs:
        f.write(stringify(o)+"\n")
```

In [32]:
```
# Save AAs & PSSMs data to different files (together makes a 3dims tensor)
# Will concat later
with open("C://Users/beshe/OneDrive/Desktop/PHYS496/MiniFold/data/input_aa.txt", "a") as f:
    for aas in input_aa:
        f.write("\nNEW\n")
        for j in range(len(aas)):
            f.write(stringify(aas[j])+"\n")
```

In [33]:
```
with open("C://Users/beshe/OneDrive/Desktop/PHYS496/MiniFold/data/input_pssm.txt", "a") as f:
    for k in range(len(input_pssm)):
        f.write("\nNEW\n")
        for j in range(len(input_pssm[k])):
            f.write(stringify(input_pssm[k][j])+"\n")
```

Done!

Implementing a Deep Learning Model (ResNet20 architecture) to Predict Dihedral Angles:



Quick Reminders:

- The ResNet is built as a 1D-ResNet and takes as input tensors of shape LxN. The window length is set to 34 and we only train and predict aangles of proteins with less than 200 AAs. No larger proteins nor crops of larger proteins are used.
- The 42 (N) channels of the input are distributed as follows: 20 for AAs in one-hot encoding (Lx20), 2 for the Van der Waals radius and the surface accessibility of the AA encoded previously and 20 channels for the Position Specific Scoring Matrix).
- We followed the ResNet20 architecture but replaced the 2D Convolutions by 1D convolutions. The network output consists of a vector of 4 numbers that represent the sin and cos of the 2 dihedral angles between two AAs (Phi and Psi).
- Dihedral angles were extracted from raw coordinates of the protein backbone atoms (N-terminus, C-alpha and C-terminus of each AA).

In [18]:
```
# Import libraries
import numpy as np
import matplotlib.pyplot as plt

# Import libraries
import keras
import keras.backend as K
from keras.models import Model

# Optimizer and regularization
from keras.regularizers import l2
from keras.losses import mean_squared_error, mean_absolute_error

# Keras layers
from keras.layers.convolutional import Conv1D
from keras.layers import Dense, Dropout, Flatten, Input, BatchNormalization, Activation
from keras.layers.pooling import MaxPooling1D, AveragePooling1D, MaxPooling2D, AveragePooling2D

# Importing Model architecture:
# Details of the model presented below:
from resnet_1d_angles import *
```

Loading the Dataset

In [19]:
```
# Load outputs/labels from file
outputs = np.genfromtxt("C:/Users/beshe/OneDrive/Desktop/PHYS496/MiniFold/data/outputs.txt")
outputs[np.isnan(outputs)] = 0.0
outputs.shape
```

Out[19]:
```
(43001, 2)
```

In [20]:
```
# Convert angles to sin/cos to remove angle periodicity
out = []
out.append(np.sin(outputs[:,0]))
out.append(np.cos(outputs[:,0]))
out.append(np.sin(outputs[:,1]))
out.append(np.cos(outputs[:,1]))
out = np.array(out).T
print(out.shape)
```

```
(43001, 4)
```

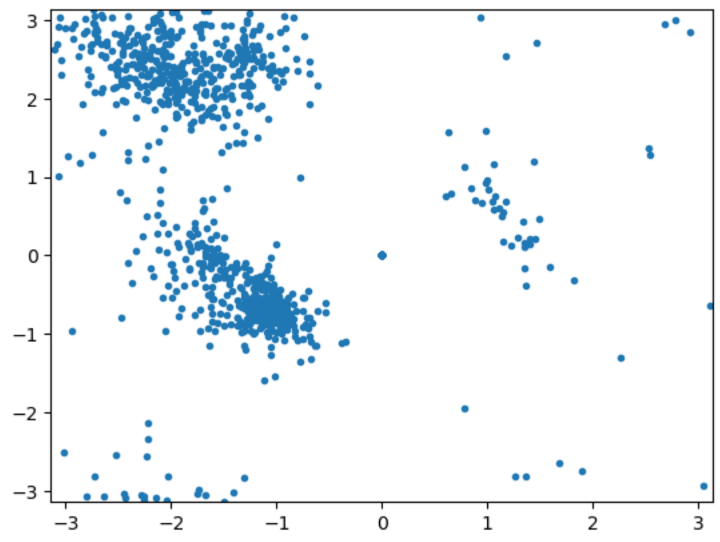
In [21]:
```
def get_ins(path = "C://Users/beshe/OneDrive/Desktop/PHYS496/MiniFold/data/input_aa.txt", pssm=None):
    """ Gets inputs from both AminoAcids (input_aa) and PSSM (input_pssm)"""
    # handles both files
    if pssm: path = "C://Users/beshe/OneDrive/Desktop/PHYS496/MiniFold/data/input_pssm.txt"
    # Opn file and read text
    with open(path, "r") as f:
        lines = f.read().split('\n')
    # Extract numeric data from text
    pre_ins = []
    for i,line in enumerate(lines):
        # Read each protein separately
        if line == "NEW":
            prot = []
            raw = lines[i+1:i+(17*2+1)]
            # Read each line as a vector + ensemble one-hot vectors as a matrix
            for r in raw:
                prot.append(np.array([float(x) for x in r.split(" ") if x != ""]))
            # Add prot to dataset
            pre_ins.append(np.array(prot))

    return np.array(pre_ins)
```

In [22]:
```
# Get inputs data
aas = get_ins()
pssms = get_ins(pssm=True)
# Check that shapes match
print(aas.shape, pssms.shape)
# Concatenate input features
inputs = np.concatenate((aas[:, :, :20], pssms[:, :, :20], aas[:, :, 20:]), axis=2)
inputs.shape
```

```
(43001, 34, 22) (43001, 34, 21)
```

Out[22]:
```
(43001, 34, 42)
```

In [23]:
```
# Plot some angle sto make sure they follow a Ramachandran plot distribution
plt.scatter(outputs[:1000,0], outputs[:1000,1], marker=".")
plt.xlim(-np.pi, np.pi)
plt.ylim(-np.pi, np.pi)
plt.show()
```



The cluster observed in the upper-left region corresponds to the angles comprised between AAs when they form a Beta-sheet while the cluster observed in the central-left region corresponds to the angles comprised between AAs when they form an Alpha-helix.

In [15]:
```
# """ WE DON'T PREPROCESS INPUTS SINCE THEY'RE IN 0-1 RANGE"""
# # Preprocess outputs (mean/std)
# # mean = np.mean(inputs,axis=(0,1,2))
# # std = np.std(inputs,axis=(0,1,2))
# # pre_inputs = (inputs-mean)/(std+1e-7)

# # print("Mean: ", mean)
# # print("Std: ", std)
```

Loading the model

In [24]:
```
# Using AMSGrad optimizer for speed
kernel_size, filters = 3, 16
adam = keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, decay=0.0, amsgrad=True)
# Create model
model = resnet_v2(input_shape=(17*2,42), depth=20, num_classes=4, conv_first=True)
model.compile(optimizer=adam, loss=custom_mse_mae,
              metrics=["mean_absolute_error", "mean_squared_error"])
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to                     
==================================================================================================
 input_1 (InputLayer)           [(None, 34, 42)]     0           []                               

 conv1d (Conv1D)                (None, 34, 16)       2032        ['input_1[0][0]']                

 batch_normalization (BatchNorm  (None, 34, 16)      64          ['conv1d[0][0]']                 
 alization)                                                                                       

 activation (Activation)        (None, 34, 16)       0           ['batch_normalization[0][0]']    

 conv1d_1 (Conv1D)              (None, 34, 16)       272         ['activation[0][0]']             

 conv1d_2 (Conv1D)              (None, 34, 16)       784         ['conv1d_1[0][0]']               

 batch_normalization_1 (BatchNo  (None, 34, 16)      64          ['conv1d_2[0][0]']               
 rmalization)                                                                                     
```

Model training

In [25]:
```
# Separate data between training and testing
split = 38700
x_train, x_test = inputs[:split], inputs[split:]
y_train, y_test = out[:split], out[split:]
```

In [26]:
```
# Resnet (pre-act structure) with 34*42 columns as inputs - leaving a subset for validation
his = model.fit(x_train, y_train, epochs=5, batch_size=16, verbose=1, shuffle=True, validation_data=(x_test, y_test))
```

```
Epoch 1/5
2419/2419 [==============================] - 66s 24ms/step - loss: 1.1790 - mean_absolute_error: 0.4631 - mean_squared_error:
0.3579 - val_loss: 1.0655 - val_mean_absolute_error: 0.4511 - val_mean_squared_error: 0.3405
Epoch 2/5
2419/2419 [==============================] - 58s 24ms/step - loss: 0.9369 - mean_absolute_error: 0.4143 - mean_squared_error:
0.3017 - val_loss: 0.8915 - val_mean_absolute_error: 0.4137 - val_mean_squared_error: 0.2987
Epoch 3/5
2419/2419 [==============================] - 47s 20ms/step - loss: 0.8250 - mean_absolute_error: 0.3918 - mean_squared_error:
0.2803 - val_loss: 0.8593 - val_mean_absolute_error: 0.4271 - val_mean_squared_error: 0.3006
Epoch 4/5
2419/2419 [==============================] - 57s 23ms/step - loss: 0.7659 - mean_absolute_error: 0.3794 - mean_squared_error:
0.2688 - val_loss: 0.8027 - val_mean_absolute_error: 0.4051 - val_mean_squared_error: 0.2921
Epoch 5/5
2419/2419 [==============================] - 55s 23ms/step - loss: 0.7316 - mean_absolute_error: 0.3725 - mean_squared_error:
0.2617 - val_loss: 0.8224 - val_mean_absolute_error: 0.4083 - val_mean_squared_error: 0.3238
```

Making predictions

In [28]:
```
preds = model.predict(x_test)
```

```
135/135 [==============================] - 2s 12ms/step
```

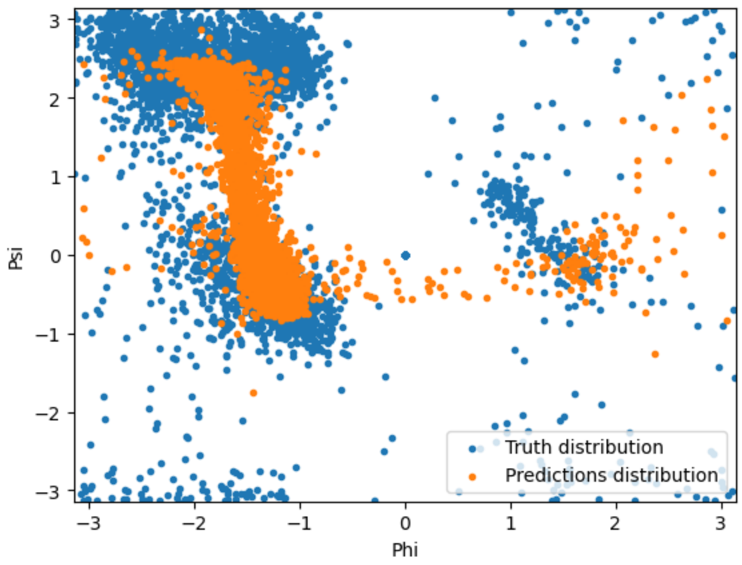
In [29]:
```
# Get angle values from sin and cos
refactor = []
for pred in preds:
    angles = []
    phi_sin, phi_cos, psi_sin, psi_cos = pred[0], pred[1], pred[2], pred[3]

    #PHI
    angles.append(np.arctan2(phi_sin, phi_cos))

    #PSI
    angles.append(np.arctan2(psi_sin, psi_cos))

    refactor.append(angles)

refactor = np.array(refactor)
print(refactor.shape)
```

```
(4301, 2)
```

In [30]:
```
# Experimental debugging prints to validate the predictions
# print("PREDS: ", preds[40:50])
# print("OUT: ", out[40:50])
# print("----------------------------------------")
# print("REFACTOR: ", refactor[:10])
# print("OUTPUTS: ", outputs[:10])
```

In [31]:
```
# Set angle range in (-pi, pi)
refactor[refactor>np.pi] = np.pi
refactor[refactor<-np.pi] = -np.pi
```

In [32]:
```
plt.scatter(outputs[split:,0], outputs[split:,1], marker=".")
plt.scatter(refactor[:,0], refactor[:,1], marker=".")
plt.legend(["Truth distribution", "Predictions distribution"], loc="lower right")
plt.xlim(-np.pi, np.pi)
plt.ylim(-np.pi, np.pi)
plt.xlabel("Phi")
plt.ylabel("Psi")
plt.show()
```



Evaluate correlation between predictions and ground truth

In [33]:
```
# Calculate Perason coefficient btwn cosines of both angles (true values and predicted ones)
cos_phi = np.corrcoef(np.cos(refactor[:,0]), np.cos(outputs[split:,0]))
cos_psi = np.corrcoef(np.cos(refactor[:,1]), np.cos(outputs[split:,1]))

print("Correlation coefficients - SOTA is Phi: 0.65 | Psi: 0.7")
print("Cos Phi: ", cos_phi[0,1])
print("Cos Psi: ", cos_psi[0,1])
```

```
Correlation coefficients - SOTA is Phi: 0.65 | Psi: 0.7
Cos Phi:  0.3929902209685417
Cos Psi:  0.3770665097029954
```

In [ ]:
```
model.save("resnet_1d_angles.h5")
```

Comments and Conclusion

The results here obtained are not as good as they could be. It's likely that the **lack of Multiple Alignmnent (MSA)**, **MSA-based features**, **Physicochemichal properties of AAs (beyond Van der Waals radius)** or the **lack of both model and feature engineering** have affected the models negatively, as well as the **little data** that they have been trained on.

For that reason, we can conclude that it has been a somehow naive approach and we expect to further implement some ideas/improvements to these models. DeepMind says: "With few or no alignments accuracy is much worse". And we have done none! But still, such an approach is more insightful, and maybe this could be used to improve results.

In [ ]:
```
```